

DATA MANAGEMENT APPLICATION PROGRAMMING INTERFACE FOR A
PARALLEL FILE SYSTEM

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the benefit of U.S. Provisional Patent Application No. 60/214,127, filed June 26, 2000. It is related to four other U.S. patent applications, filed on even date, entitled "Data Management Application Programming Interface Session Management for a Parallel File System"; "Implementing Data Management Application Programming Interface Access Rights in a Parallel File System"; "Data Management Application Programming Interface Handling Mount on Multiple Nodes in a Parallel File System"; and "Data Management Application Programming Interface Failure Recovery in a Parallel File System." All of these related applications are assigned to the assignee of the present patent application and are incorporated herein by reference.

FIELD OF THE INVENTION

The present invention relates generally to computer file systems, and specifically to implementation of data management applications in parallel file systems.

BACKGROUND OF THE INVENTION

A wide variety of data management (DM) applications have been developed to supplement the basic file storage and retrieval functions offered by most operating system (OS) kernels. Typical DM applications include hierarchical storage management (also known as data migration), unattended data backup and recovery, on-line encryption and compression, and directory browsers.

These applications, which extend the basic OS kernel functions, are characterized by the need for monitoring and controlling the use of files in ways that ordinary user applications do not require.

5 In response to this need, the Data Management Interfaces Group (DMIG) was formed by a consortium of UNIX® software vendors to develop a standard Data Management Application Programming Interface (DMAPI). DMAPI provides a consistent, platform-independent
10 interface for DM applications, allowing DM applications to be developed in much the same way as ordinary user applications. By defining a set of standard interface functions to be offered by different OS vendors, DMAPI gives DM software developers the tools they need for
15 monitoring and controlling file use, without requiring them to modify the OS kernel. DMAPI is described in detail in a specification document published by the Open Group (www.opengroup.org), entitled "Systems Management: Data Storage Management (XDSM) API" (Open Group Technical
20 Standard, 1997), which is incorporated herein by reference. This document is available at www.opengroup.org.

As noted in the XDSM specification, one of the basic foundations of DMAPI is "events." In the event paradigm,
25 the OS informs a DM application running in user space whenever a particular, specified event occurs, such as a user application request to read a certain area of a file. The event may be defined (using DMAPI) as "synchronous," in which case the OS will notify the DM
30 application of the event and will wait for its response before proceeding, or as "asynchronous," in which case OS processing continues after notifying the DM application

of the event. The area of a file with respect to which certain events are defined is known as a "managed region."

Another fundamental concept in DMAPI is a "token,"
5 which is a reference to a state that is associated with a synchronous event message. The state typically includes lists of files affected by the event and DM access rights in force for those files. The token may be passed from thread to thread of the DM application and provides a
10 convenient means for referencing the state. Access rights may either be shared with other processes (in which case they are read-only rights), or they may be exclusive (read-write) rights.

Communications between DM applications and the OS
15 are session-based. The DM application creates the session by an appropriate DMAPI function call (`dm_create_session()`). The application then registers event dispositions for the session, indicating which event types in a specified file system should be
20 delivered to the session. Multiple sessions may exist simultaneously, and events in a given file system may be delivered to any of these sessions.

The DMAPI standard, having grown out of the needs of UNIX system vendors, is based on the notion of a single
25 system environment, using a single computing node. DMAPI implementations have also been developed for distributed file systems, which allow a user on a client computer connected to a network to access and modify data stored in files on a file server. When a user accesses data on
30 the file server, a copy of the data is stored, or cached, on the client computer, and the user can then read and modify the copy. When the user is finished, the modified

data are written back to the file server. Examples of distributed file systems include Sun Microsystems' Network File System (NFS™), Novell Netware™, Microsoft's Distributed File System, and IBM/Transarc's DFS™.

5 Transarc Corporation (Pittsburgh, Pennsylvania) has developed a DMAPI implementation for its DFS called DMEpi. All of these distributed file systems, however, are still essentially single-node systems, in which a particular server controls any given file. The DMAPI and
10 data management applications for such distributed file systems are essentially server functions and are not distributed among the client nodes.

IBM's General Parallel File System (GPFS) is a UNIX-style file system designed for IBM RS/6000
15 multiprocessor computing platforms, such as the SP™ and HACMP™ systems. GPFS, which runs in the AIX® operating system, allows applications on multiple nodes to share file data, without mediation of a file server as in distributed file systems. GPFS is described, for
20 example, in a publication entitled "General Parallel File System for AIX: Concepts, Planning and Installation," which is available at www.rs6000.ibm.com/resource/aix_resource/sp_books/gpfs. GPFS supports very large file systems and stripes data across multiple disks for
25 higher performance. GPFS is based on a shared disk model that provides low-overhead access to disks not directly attached to the application nodes and uses a distributed locking protocol to provide full data coherence for access from any node. These capabilities are available
30 while allowing high-speed access to the same data from all nodes of the system. GPFS has failure recovery

39878S5

capabilities, allowing applications to continue running even when node or network component failures occur.

A series of patents to Schmuck et al. describe aspects of a shared parallel disk file system that are
5 implemented in GPFS. These patents include U.S.
5,893,086; U.S. 5,940,838; U.S. 5,963,963; U.S.
5,987,477; U.S. 5,999,976; U.S. 6,021,508; U.S.
6,023,706; and U.S. 6,032,216, all of whose disclosures
are incorporated herein by reference.

5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200

SUMMARY OF THE INVENTION

Preferred embodiments of the present invention provide a DMAPI that is suitable for use in a multi-node, parallel computing environment, and specifically for use with parallel file systems. Implementing DMAPI in a parallel file system, such as the above-mentioned GPFS, requires enhancements to the functions defined in the XD SM standard and alterations in certain basic definitions and assumptions that underlie DMAPI implementations known in the art. The basic semantics and functionality of the standard DMAPI model, however, are preferably preserved in the parallel system. DM application programmers are thus enabled to integrate data migration and other DM applications with the parallel file system in an immediate and straightforward manner.

In preferred embodiments of the present invention, computing nodes in a cluster are mutually linked by a suitable interconnection to a set of one or more block storage devices, typically disks. A parallel file system is configured so that all nodes in the cluster can mount the same set of file system instances. File data and metadata, on multiple logical volumes, may reside at different nodes. All of the volumes are accessible from all of the nodes via a shared disk mechanism, whereby the file data can be accessed in parallel by multiple tasks running on multiple nodes. The enhanced DMAPI provided for the parallel file system is used to support DM functions, such as automatic data migration, over all of the nodes and storage volumes in the cluster.

DM applications may run on substantially any of the nodes in the cluster, as either single-node or multi-node

parallel applications. The DM application preferably starts by creating a session on one of the nodes and specifying the DM events that are to be reported to the session. The node on which the session is created is
5 designated as the session node, and all specified events generated by file system operations are reported to the session node, regardless of the node at which the events are generated. Thus, an event may be generated by a file operation on one of the nodes, referred to herein as the
10 source node, and delivered to a session on a different node, i.e., the session node. If the event is a synchronous event, requiring a response from the DM application before the file operation can continue, the source node will wait to carry out the requested file
15 operation until the session node has sent its response back to the source node. In contrast, in DMAPI implementations known in the art all events and sessions take place on a single node.

There is therefore provided, in accordance with a
20 preferred embodiment of the present invention, in a cluster of computing nodes having shared access to one or more volumes of data storage using a parallel file system, a method for managing the data storage, including:

25 initiating a session of a data management (DM) application on a first one of the nodes;

running a user application on a second one of the nodes;

30 receiving a request submitted to the parallel file system by the user application on the second node to perform a file operation on a file in one of the volumes of data storage; and

sending a DM event message from the second node to the first node responsive to the request, for processing by the data management application on the first node.

Preferably, initiating the session includes
5 initiating the session in accordance with a data management application programming interface (DMAPI) of the parallel file system, and receiving the request includes processing the request using the DMAPI. Further preferably, receiving and processing the event message at
10 the first node using one or more functions of the DMAPI called by the data management application. Additionally or alternatively, sending the event message includes sending the message for processing in accordance with a disposition specified by the data management application
15 using the DMAPI for association with an event generated by the file operation.

Preferably, the method further includes receiving a response to the event message from the data management application on the first node, and performing the file
20 operation requested by the user application on the second node subject to the response from the data management application on the first node. Further preferably, receiving the request includes submitting the request using a file operation thread running on the second node,
25 and blocking the thread until the response to the event message is received from the first node. Additionally or alternatively, sending the event message includes passing the event message from a source physical file system (PFS) on the second node to a session PFS on the first
30 node, and receiving the response includes passing a response message from the session PFS to the source PFS.

In a preferred embodiment, the method includes receiving the event message at the first node, obtaining a data management access right from a physical file system (PFS) at the first node responsive to the event
5 message, and processing the event message using the access right.

In a further preferred embodiment, receiving the request includes receiving first and second requests of different types submitted to a physical file system (PFS)
10 at the second node, wherein based on the different request types, sending the event message includes sending a first event message to the first node responsive to the first request, and sending a second event message responsive to the second request to a further node, on
15 which a further data management application session has been initiated. Preferably, sending the first and second event messages includes receiving at the second node a specification of event types and their respective dispositions, the event types corresponding to the
20 requests to perform the file operations, and dispositions indicating which of the event messages should be sent to which of the nodes, and sending the messages responsive to the specification.

In yet a further preferred embodiment, running the
25 user application includes running a first user application instance on the second node, and running a further user application instance on a further one of the nodes, and the method includes receiving a further request submitted to the parallel file system by the
30 further user application instance to perform a further file operation, and sending a further event message responsive to the further request for processing by the

data management application on the first node. Optionally, the further one of the nodes is the first node.

5 In still a further preferred embodiment, initiating the session of the data management application includes initiating a data migration application, so as to free storage space on at least one of the volumes of data storage.

10 Preferably, the method includes choosing one of the nodes to act as a session manager node, wherein initiating the session includes sending a message from the session node to the session manager node, causing the session manager node to distribute a specification of events and respective dispositions of the events for the
15 session among the nodes in the cluster, and wherein sending the DM event message includes sending the message in accordance with the dispositions.

In a preferred embodiment, one of the nodes is appointed to serve as a respective file system manager
20 for each of one or more file systems in the cluster, and for each of the file systems, the session manager node sends the specification of the dispositions applicable to the file system to the respective file system manager, which sends the dispositions to all of the nodes in the
25 cluster on which the file system is mounted.

Preferably, sending the DM event message includes incorporating in the message a data field uniquely identifying the second node.

30 Further preferably, upon receiving from one of the nodes other than the first one of the nodes a call for a data management application programming interface (DMAPI) function in connection with the session, the function is

performed only if it does not change a state of the session or of an event associated with the session.

There is also provided, in accordance with a preferred embodiment of the present invention, computing apparatus, including:

one or more volumes of data storage, arranged to store data; and

a plurality of computing nodes, linked to access the volumes of data storage using a parallel file system, and arranged so as to enable a data management (DM) application to initiate a data management session on a first one of the nodes, while allowing a user application to run on a second one of the nodes, so that when the user application submits a request to the parallel file system on the second node to perform a file operation on a file in one of the volumes of data storage, a DM event message is sent from the second node to the first node responsive to the request, for processing by the data management application on the first node.

There is additionally provided, in accordance with a preferred embodiment of the present invention, a computer software product for use in a cluster of computing nodes having shared access to one or more volumes of data storage using a parallel file system, the product including a computer-readable medium in which program instructions are stored, which instructions, when read by the computing nodes, cause a session of a data management (DM) application to be initiated on a first one of the nodes, while allowing a user application to run on a second one of the nodes, and in response to a request submitted to the parallel file system by the user application on the second node to perform a file

39878S5

operation on a file in one of the volumes of data storage, cause the second node to send a DM event message to the first node, for processing by the data management application on the first node.

- 5 The present invention will be more fully understood from the following detailed description of the preferred embodiments thereof, taken together with the drawings in which:

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram that schematically illustrates a cluster of computing nodes with a parallel file system, in accordance with a preferred embodiment of the present invention;

Fig. 2 is a block diagram that schematically shows details of the parallel file system of Fig. 1, in accordance with a preferred embodiment of the present invention;

Fig. 3 is a flow chart that schematically illustrates a method for handling a DMAPI event generated by a file operation in a parallel file system, in accordance with a preferred embodiment of the present invention;

Fig. 4 is a flow chart that schematically illustrates a method for unmounting an instance of a parallel file system, in accordance with a preferred embodiment of the present invention;

Fig. 5 is a flow chart that schematically illustrates a method for handling a DMAPI function call in a parallel file system, in accordance with a preferred embodiment of the present invention; and

Fig. 6 is a flow chart that schematically illustrates a method for handling a DMAPI session failure in a parallel file system, in accordance with a preferred embodiment of the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

GLOSSARY

The following is a non-exhaustive list of technical terms that are used in the present patent application and in the claims. The list is provided here for the convenience of the reader. Certain of the items in the list are specific to preferred embodiments of the present invention. These terms are described at greater length in the Detailed Description following the Glossary.

- 10 • **Cluster:** A collection of computers interconnected by a communication mechanism, typically a high-performance switch or a network. The computers in the cluster collaborate with each other in computations and share data resources.
- 15 • **Node:** A computer that is part of a cluster. Each node in the cluster has a cluster-wide unique identity.
- 20 • **File System:** A hierarchical collection of files and file directories that are stored on disk and have an identified root, and are accessed using a predefined interface. Typically, such interfaces follow the prescription of standards, such as Posix. The term "file system" is also loosely used to describe the data and metadata contained in the file system.
- 25 • **File System Instance:** A file system that is mounted on a computer. In a cluster, a given file system can have multiple file system instances, each instance mounted on a different node.

- **Physical File System (PFS):** A software component that manages collections of data on disks, typically using mechanisms and interfaces prescribed by the X/Open and Posix standards. The PFS is one of
5 layers in the hierarchy of interfaces that are used to support the file system and to enable software applications to access the file data. Multiple different PFSs may coexist on a computer, each used to implement a different type of file system. The
10 PFS usually runs in the kernel, with possible extensions running as daemons in user space.
- **Parallel File System:** A PFS running on a cluster of nodes, which enables all nodes in the cluster to access the same file data concurrently. Preferably,
15 all nodes share in the management of the file systems. Any of the nodes can perform any role required to manage the file systems, with specific roles assigned to particular nodes as needed. The term "parallel file system" is also loosely used to describe a file system that is managed by a PFS
20 software component, as defined in this paragraph.
- **DMAPI:** Data Management Application Programming Interface, as specified in the above-mentioned XDSM standard. This term is also used to denote the
25 software sub-component implementing the interface in the PFS.
- **Session Node:** A node in a cluster on which one or more data management (DM) sessions have been created. (The term "data management" is used as
30 defined in the XDSM standard.) The term is also

used to identify a specific node at which a specific session exists.

- 5 • **Source Node:** A node in a cluster that generates DM events. (The term "events" is similarly used as defined in the XD SM standard.) The term is also used to identify a specific node that generated a specific event.
- 10 • **Session Manager (SM):** A node in a cluster that is assigned the role of coordinating the creation and maintenance of DM sessions and DM event dispositions on the nodes in the cluster. Typically, the sessions may be created on any of the nodes, including the session manager node.
- 15 • **File System Manager (FSM):** A node in a cluster that is assigned the role of managing the metadata of a specific file system. The node coordinates among all the instances of the file system that are mounted on the different nodes in the cluster.
- 20 • **Persistent Data:** Data used by a software component on a computer or a cluster of nodes, which is not lost when the component crashes. Persistent data can be recovered and restored when the component recovers.
- 25 • **Single Node Failure:** A failure that occurs on a single node in a cluster. Services provided by other nodes in the cluster are normally not affected by this failure. The term is specifically used to describe the failure of the PFS software component. All file systems managed by the failing PFS become
30 inaccessible on the failing node, but typically remain accessible on other nodes in the cluster.

- **Total PFS Failure:** Failure of the PFS software component on all the nodes in a cluster. All file systems managed by the failing PFS become inaccessible on all nodes in the cluster.
- 5 • **File System Failure:** Failure of a file system instance on a single node, possibly due to disconnection from disks or from other nodes in a cluster. The failing file system instance becomes inaccessible on the node where the failure occurred.
- 10 The term is also used to describe the failure of multiple (or all) instances of the same file system on any node in the cluster.
- **Session Failure:** Failure of the PFS on a session node, at which one or more sessions exist for monitoring file systems managed by the failing PFS.
- 15 Such sessions can no longer be used, and are hence called failed sessions. When the PFS recovers, the sessions can be recreated. The term is also used to identify the failure of a specific session on a
- 20 specific node.

SYSTEM OVERVIEW

Fig. 1 is a block diagram that schematically illustrates a cluster 20 of computing nodes 22, running a parallel file system, identified as a physical file system (PFS) 28, in accordance with a preferred embodiment of the present invention. Nodes 22 are connected to one another and to multiple disks 24 by a suitable communication mechanism, such as a high-capacity switch or network (not shown). The disks may thus be

25 accessed in parallel by any of the nodes. Preferably,

30

cluster 20 comprises an IBM SP or HACMP system, and PFS 28 comprises GPFS, running on the AIX operating system, as described in the Background of the Invention. Alternatively, nodes 22 may be arranged in other cluster configurations, and PFS 28 may comprise other types of parallel file systems and operating platforms. In this regard, it is noted by way of example that the IBM GPFS file system has been ported to other platforms, such as Linux.

10 All nodes 22 in cluster 20 can mount the same file systems. The file data and metadata of the file systems are striped across multiple logical volumes, each of which may reside on a different node 22. All of the volumes are accessible from all of the nodes via a shared disk mechanism, preferably the virtual shared disk (VSD) mechanism used by GPFS. File data can thus be accessed in parallel by multiple tasks on multiple nodes. Parallel access is preferably protected by a byte range lock, as described in the above-mentioned patents by Schmuck et al.

20 One of nodes 22, is selected to serve as cluster configuration manager (CM) 34. If the CM fails, another node takes its place. The CM performs various global coordination duties in cluster 20. These duties include appointing one of the nodes to serve as file system manager (FSM) 36 for each file system in the cluster. The FSM serves all of the nodes in the cluster that mount the given file system. It is responsible for metadata management, which is used to maintain the file system instances on all of the nodes in a consistent state.

30 Nodes 22 are capable of running both user applications 30 and data management (DM) applications 32.

These applications may be either single-node or parallel, multi-node applications, which take advantage of PFS 28 to access disks 24. A Data Management Application Programming Interface (DMAPI) 26 is preferably integrated into physical file system (PFS) 28. DM applications 32 use DMAPI 26 to track and control file operations and to manage file data of file systems in cluster 20, as described in detail hereinbelow. For this purpose, DMAPI 26 uses mechanisms and infrastructure provided by the PFS, including communication, memory management, locking and synchronization. In a similar vein, configuration manager 34 of PFS 28 also serves as a session manager (SM) for DMAPI 26.

PFS 28 with DMAPI 26 is typically supplied as a software package for installation on cluster 20, with or without a complete operating system, such as AIX. This software may be downloaded to the cluster in electronic form, over a network, for example, or it may alternatively be supplied on tangible media, such as CD-ROM, for installation on the cluster nodes.

Fig. 2 is a block diagram that schematically shows further details of the software structure and operation of PFS 28, and particularly of DMAPI 26, in accordance with a preferred embodiment of the present invention. The figure shows a session node 40, a source node 42 and a session manager (SM) node 44, all of which are nodes 22 (Fig. 1) in cluster 20. DM application 32 is shown to be running on node 40, while user application 30 is running on node 42. Typically, both DM application 32 and user application 30 may be distributed applications, running on multiple nodes simultaneously, and possibly running together on the same node. For the sake of simplicity of

illustration, however, and without loss of generality, only a single node of each type is shown in Fig. 2. Similarly, although SM 34 is shown as running on a separate session manager node 44, it may alternatively
5 run on any of the nodes in cluster 20, including session node 40 and source node 42.

In an alternative embodiment of the present invention, not shown in the figures, DM sessions are explicitly replicated on all nodes in the cluster. DM
10 events generated at a source node are then delivered to a session at the source node itself. This embodiment requires that DM applications be defined as multi-node, parallel applications, unlike DM applications known in the art. Each event is handled by the DM application
15 instance on the source node at which it originated, while consistency is maintained among the instances using methods known in the art of parallel applications.

SESSIONS AND EVENT HANDLING

Upon initiation of DM application 32, the
20 application creates a DMAPI session on session node 40. Dispositions 49 of enabled events 46 are maintained on source node 42 and on SM node 44. A list of enabled events can be associated individually with a file and globally with an entire file system. Conflicts between
25 individual and global event lists are preferably resolved in favor of the individual list. Preferably, event lists are persistent and are kept with the file system in stable storage. Dispositions are not persistent and must be set explicitly for each file system after PFS 28 is
30 started and sessions are created.

When user application 30 on source node 42 invokes a file operation that generates an enabled event having a disposition in disposition list 49, DMAPI 26 sends the event to an event queue on session node 40. (Node 42 is referred to as the source node, because it is the source of the generated event in the present example.) When the event is a synchronous event, PFS 28 on the source node waits for the response of the DM application before allowing the file operation to proceed. The process of sending events and responses is described in greater detail hereinbelow with reference to Fig. 3.

In practice, multiple DM applications and multiple sessions may run simultaneously in cluster 20. Each session can monitor multiple file systems. At the same time, multiple DM application threads can access the same file in parallel. Moreover, a given file system may be monitored by multiple sessions, created by the same DM application or by different DM applications running concurrently, on the same node or on different nodes. When there are multiple sessions monitoring a given file system, event dispositions are partitioned among the sessions by event type, so that there is no ambiguity as to which session will handle each type of event. On the other hand, multiple sessions can also be used to monitor different, respective file systems.

DM application 32 can set the disposition of events or change the disposition of events in a given file system from one session to another, as noted above, using the DMAPI `dm_set_disp()` function. This call can come from any node at substantially any time. The node initiating the change notifies SM 34, which accordingly updates dispositions 49 in its own, centrally-maintained

39878S5

data structures. The SM sends the changes to the relevant FSM 36, which then sends the information to all nodes on which the file system is mounted. When a new FSM is appointed (at the first mount of a given file system, or after failure of the previous FSM), it obtains the event dispositions from SM 34. When additional nodes mount the file system, they obtain the dispositions from the relevant FSM. This approach guarantees that the FSM and all nodes mounting the file system will always have the most current event dispositions, while maintaining efficiency of communications by disseminating information only to the nodes for which it is relevant.

SM 34 is responsible for coordinating among the session nodes, in order to ensure that all session identifiers (IDs) are unique and that event disposition partitioning is consistent. All nodes 22 in cluster 20 know the identity of the session manager.

A new session is created by invoking the DMAPI function `dm_create_session()` on the session node, which then sends a message to SM 34. The SM generates a session ID, adds the session to its list of all sessions in the cluster, broadcasts the session details to all nodes in the cluster, and returns the session ID to the session node. The session becomes valid only after session node 40 and SM 44 complete all of these messages. Session IDs are preferably unique over time in the cluster, most preferably consisting of a time stamp and the node ID of SM 34. The use of globally-unique session IDs, which are never reused, prevents naming consistency problems in the multinode environment.

A new session node assumes an existing session by invoking `dm_create_session()`, specifying the existing

39878S5

session ID. DMAPI 26 sends the ID to SM 34, which updates the session details in its list. An existing session will be assumed in this manner only after a session node failure, as described below. The session ID
5 does not change when an existing session is assumed. `dm_create_session()` can also be used to modify the session information string of an existing session. This call can be made only on the session node.

A session can register (or unregister) to receive an
10 event when a file system is mounted on one of nodes 22 by calling the DMAPI function `dm_set_disp()` on the session node. DMAPI 26 informs SM 34, and the SM keeps track of the sessions that are registered for the mount event. When a node performs a mount operation, it obtains from
15 the SM a list of sessions that are registered for the mount event.

A session may be destroyed by invoking the DMAPI function `dm_destroy_session()` on session node 40. In this case, the session node sends a message to SM 34,
20 which removes the session from its list and broadcasts the change to all nodes.

For efficient event generation and recovery from PFS failure, the session and event information is replicated on multiple nodes in cluster 20. Preferably, as shown in
25 Fig. 2, information regarding each outstanding event 46 is maintained on both session node 40 and on source node 42. Session data 48 are maintained on session node 40 and on session manager node 44. Preferably, every node in the cluster maintains partial information on every
30 session in the cluster, including session ID, session node address, whether or not the session is registered for the mount event, and a short prefix of the session

information string. Dispositions 49 are maintained by SM 34 on node 44, and on all nodes on which the monitored file system is mounted, including source node 42.

5 SM 34 is responsible for disseminating changes in session details. When the SM is notified of a change in the state of one of the sessions, it broadcasts the change to all of the nodes in the cluster.

10 Replication of the session information and event dispositions on multiple nodes allows DMAPI events to be generated efficiently, without repeatedly communicating session and disposition information between nodes. It also supports efficient recovery from single node failure without the necessity of using stable storage. Methods for handling session failures in system 20 are described
15 hereinbelow with reference to Fig. 6.

Fig. 3 is a flow chart that schematically illustrates a method for handling a DMAPI event generated in cluster 20, in accordance with a preferred embodiment of the present invention. The method begins when user
20 application 30 invokes a file operation on source node 42, at an invocation step 50. It is assumed that this file operation generates an event that appears in the list of enabled events 46 and dispositions 49, which are furnished by SM 44 to all nodes in system 20 that have
25 mounted instances of the file system in question. Typically, if the event is enabled, but there is no disposition listed, DMAPI 26 will return an error message to the user application.

Event generation is preferably implemented in the
30 virtual file system (VFS) interface layer of PFS 28. The definition of the VFS layer and its interaction with the PFS are well known in the art of UNIX-type operating

systems, including the above-mentioned IBM AIX operating system. The integration of DMAPI 26 with PFS 28 includes augmenting the file operations in the PFS with code for event generation. In an event generation step 52, this
5 code causes the file operation client thread invoked at step 50 to generate the prescribed event. If the event is an asynchronous event, PFS 28 on source node 42 sends an appropriate event message to the session on session node 40, and the requested file operation is then
10 immediately free to proceed on source node 42. In the example shown in Fig. 3, however, the event is assumed to be a synchronous event, which causes the file operation thread to block and await a response from the DM application before proceeding with PFS processing.

15 PFS 28 on source node 42 sends an event message to PFS 28 on session node 40, in accordance with the specified event disposition, at an event sending step 54. The event message header preferably carries a field, `ev_nodeid`, which is added to the `dm_eventmsg` structure
20 defined in the above-mentioned XDASM specification in order to identify source node 42 (since events can be generated at any node in cluster 20). In implementations based on GPFS in the SP environment, the node identifier is preferably its System Data Repository (SDR) node
25 number. The event message is enqueued at session node 42.

DM application 32 on session node 40 receives and handles the event sent by PFS 28, at an event handling step 56. For this purpose, the DM application makes use of function calls provided by DMAPI 26, such as
30 `dm_get_events()`, as specified by the XDASM standard. These function calls are implemented as kernel calls from the user space of the DM application into PFS 28, based

39878S5

on linking the DM application with an interface library of DMAPI function calls, as is known in the art. DM function calls enter the PFS kernel on the DM application thread. The processing may involve additional PFS daemon
5 threads, and may proceed both in user and kernel space.

After DM application 32 has processed the event, it generates its response to the event, at a response step 58, using the DMAPI function call `dm_respond_event()`. Session node 40 sends the response back to source node
10 42, at a response sending step 60. The PFS on node 42 passes the event response to the file operation thread, at a response reception step 62. If the response indicates that the operation should be aborted, the file operation returns to user application 30 without further
15 PFS processing; otherwise, the file operation continues its PFS processing until completion, at a continuation or aborting step 64.

MOUNTING AND UNMOUNTING FILE SYSTEM INSTANCES

PFS 28 generates a mount event each time a mount
20 operation is performed on one of the nodes in cluster 20. Similarly, each unmount operation on any node generates preunmount and unmount events, assuming such events are enabled and have a disposition. DM application 32 should therefore be capable of handling multiple mount,
25 preunmount and unmount events, corresponding to multiple instances of the file system that are mounted on multiple nodes. By contrast, in single-node systems, as implied by the XDMS standard, DM applications are not required to deal with more than a single mount, preunmount or unmount
30 event per file system. As a result, in single-node

systems, the preunmount and unmount events always follow the mount event in serial order.

In PFS 28, on the other hand, there is not a predictable serial relationship between all of the mount, preunmount and unmount events of each file system. Without serialization of all mount and unmount operations of the file system, there is no practical way to designate the first or last mount or unmount. There need not even be a match between the number of mount events and the number of preunmount or unmount events for a given file system, since an unmount that is initiated internally by PFS 28 (due to forced unmount or PFS shutdown, for example) will not generate any events. Therefore, DMAPI 26 requires methods for handling mount, preunmount and unmount events that extend those provided by the XDSM standard.

To provide additional information to DM application 32, two new flags, DM_LOCAL_MOUNT and DM_REMOTE_MOUNT, not defined in the XDSM standard, are preferably added to the mode fields in the message structures of mount, preunmount and unmount events (the me_mode and ne_mode fields, respectively). When DM_LOCAL_MOUNT is set, the mount or unmount operation concerned is local to the session node. When DM_REMOTE_MOUNT is set, the operation is at a node that is remote from the session node. In this case, the ev_nodeid field mentioned above can be used by the session node to identify the source node on which the mount or unmount operation is to be performed. These flags are also used in the dm_mount_event data structure returned by the DMAPI function dm_get_mountinfo(). This function can be called from any node, even if the file system is not mounted on that

node. At least one of the two flags will be set in the data structure that is returned, as long as the file system is mounted on one or more of the nodes in cluster 20.

5 DM application 32 can make good use of the enhanced node information provided by the `dm_get_mountinfo()` function for processing of mount and preunmount events. For example, before the DM application responds to a mount event received from a node that is not the session
10 node, it can invoke `dm_get_mountinfo()` to determine whether the relevant file system is already mounted locally at the session node. If not, the DM application preferably performs a local mount.

Mount events are preferably enqueued in the session
15 queue ahead of other events, in order to improve the response time of PFS 28 to mount operations when the queue is busy.

Fig. 4 is a flow chart that schematically illustrates a method for performing an unmount operation
20 on source node 42, in accordance with a preferred embodiment of the present invention. This method is used whenever an unmount operation is invoked, at an unmount invocation step 70. DMAPI 26 on source node 42 generates a preunmount event message and waits for the response
25 from the DM application, at an event generation step 72. This message is received and processed by DMAPI 26 on session node 40. DMAPI 26 on source node 42 receives the response from session node 40, at a response receipt step 74.

30 DMAPI 26 checks whether the unmount operation is a forced unmount, at a force checking step 76. In such a case, any outstanding access rights for the relevant

39878S5

file system on source node 42 are released, at a cleanup step 78. DMAPI 26 then permits the unmount to proceed, at an unmounting step 80.

On the other hand, if the unmount is not a forced
5 unmount, DMAPI 26 checks whether it has received an
"abort" response from session node 40, at an abort
checking step 82. If so, the unmount operation is
failed. Similarly, the unmount is failed if there are
still DM access rights to the file system on source node
10 42, at an access right checking step 84. In either of
these cases, an error code is set for the unmount
operation on node 42, at an error step 86. Only if there
are no outstanding access rights on node 42 can the
unmount proceed normally at step 80.

15 Whether the unmount was performed successfully (step
80) or not (step 86), DMAPI 26 generates an unmount event
and waits for a response, at an unmount event generation
step 88. After receiving the response, any error code
that was set at step 86 is returned by the unmount
20 operation, at an error return step 89.

DM ACCESS RIGHTS

DM applications acquire DM access rights to file
objects and associate them with event tokens. DM access
rights are required in some of the DMAPI functions
25 specified in the XD SM standard (which are implemented in
the multi-node environment by the present invention). In
order to avoid overhead that would be incurred by
managing access rights distributively in the cluster
setting, all of the access rights associated with a given
30 event are preferably managed by the corresponding session
node. Thus, all requests to acquire, change, query or

39878S5

release DM access rights must be invoked by the DM application on session node 40.

File operations must abide by DM access rights. In particular, file operations that conflict with DM access rights must be blocked while the access is held by the DM application. Conversely, the DM application must be prevented from acquiring an access right while a conflicting file operation is in progress. Preferably, these access rights are implemented using the internal locking mechanisms of PFS 28, such as the GPFS locking mechanisms described in the above-mentioned patents by Schmuck et al.

DM access rights in cluster 20 are preferably treated as an additional file lock in the hierarchy of locks acquired during file access. This approach enables acquiring and releasing access rights efficiently, using existing, highly-optimized locking mechanisms of the PFS. This additional lock is referred to herein as the "DM lock." The lock characteristics are affected by the type of access (shared or exclusive) and the type of thread acquiring the lock (file operation thread or data management operation thread). Existing file locks (such as those described by Schmuck et al.) cannot be used for this purpose, since DM access rights are held across multiple kernel calls and can be shared among DM application threads without going through the kernel. The existing file locks are still required to synchronize access to file data, even while a DM access right is held. Preferably, to prevent deadlocks, the DM lock is acquired before any other locks in the file locking hierarchy.

39878S5

Table I below is a lock conflict table that defines DM access rights semantics in cluster 20. Four lock modes are used:

- DMX - data management exclusive access.
- DMS - data management shared access.
- FSX - file system exclusive access.
- FSS - file system shared access.

An "X" in the table indicates a conflict between the corresponding modes.

TABLE I - DM ACCESS RIGHTS

	FSS	FSX	DMS	DMX
FSS				X
FSX			X	X
DMS		X		X
DMX	X	X	X	X

DMX and DMS modes are used only in DM operations. They provide exclusive and shared access rights, respectively, for each individual DM operation, as defined by the XDSM standard.

FSX and FSS modes are used in file operations, in order to prevent DM applications from acquiring a DM access right while a conflicting file operation is in progress. FSX prevents acquisition of any DM access rights. FSS prevents acquisition of exclusive DM access rights, but does not conflict with shared DM access rights. Typically, a file operation that modifies the data in a file object or destroys the object will acquire a FSX lock, whereas a FSS lock will suffice for other file operations. There is no conflict between the FSX

39878S5

and FSS modes, because file operations never compete with one another for DM locks. This feature is important in reducing the performance impact of the additional DM lock in parallel file systems, since locking conflicts in such systems are resolved by communication among multiple nodes.

In the GPFS parallel file system, as described by Schmuck et al., file locking is implemented using a token manager, which grants lock tokens to nodes upon request and revokes them when other nodes make conflicting requests. For the DM lock, after the token manager grants a FSX or FSS token to a node, there will be no need to revoke the token until a DM application requests a DM lock on the file. For files being used actively by a user application, interference by typical DM applications is generally expected to be minimal. The added overhead in normal file operations that is associated with DM lock acquisitions and revocations should therefore be small.

While a file operation holds its DM lock only for the duration of the operation (a single kernel call), a DM application can hold its DM lock across many DMAPI function calls (multiple kernel calls). This continuous holding of the DM lock is achieved by associating the access right in question with an event token, which can be presented in subsequent DMAPI calls.

DMAPI functions can be invoked without presenting a DM token (DM_NO_TOKEN), in which case DM access rights are acquired only for the duration of the given DMAPI function. For this purpose, a FSS or FSX lock is sufficient, instead of the more restrictive DMS or DSM

39878S5

lock. The tokenless DMAPI call thus uses the same type of DM locking as do regular file operations.

INVOKING DMAPI FUNCTIONS IN A CLUSTER

In preferred embodiments of the present invention,
5 in the multi-node environment of cluster 20, DMAPI functions can be called from any of the nodes in the cluster. Functions that do not change the state of a session or event can be invoked freely at any node, in order to enable DM applications to exploit the inherent
10 parallelism of the PFS. `dm_punch_hole()` and `dm_read_invis()` are examples of such non-state-changing DMAPI functions. On the other hand, DMAPI functions that change the state of a session or event must be invoked on the session node.

15 Fig. 5 is a flow chart that schematically illustrates a method for processing a DMAPI function call that provides as parameters a session ID and event token, in accordance with a preferred embodiment of the present invention. This method is invoked when DMAPI 26 receives
20 a function call from DM application 32 on node 22, at a function calling step 90. DMAPI 26 ascertains whether node 22 is the session node (such as node 40) for the session, at a session node determination step 92. If so, the required function is carried out by PFS 28, in a
25 function performing step 100.

If the node invoking the function call is not the session node for this session, DMAPI 26 determines whether this function changes the state of the DM event or session (as specified by the XD SM standard), at a
30 state change determination step 96. If so, the requested

39878S5

function is failed, and DMAPI 26 returns an error to DM application 32, at a failure step 98.

On the other hand, if this is not the session node, and the DMAPI function does not change the current event
5 or session state, the call proceeds in step 100 as long as the session exists on some node, and the required event token is presented. Optionally, the DM application on the requesting node caches a copy of the token.

DMAPI FAILURE AND RECOVERY MECHANISMS

10 The failure model defined in the XDSM standard is geared to a single-node system, in which two types of DMAPI-related failures may occur: DM application failure or total PFS failure. When only the DM application fails, DMAPI resources, such as sessions and events,
15 remain intact. As a result, file systems may become unstable, since there may be pending events and blocked user threads, waiting for response by the DM application. To deal with this situation, the DM application must restart and pick up any existing session where it left
20 off. For this purpose, the XDSM standard provides DMAPI functions that enable the restarted DM application to query the session queue and handle any pending events.

Recovery from total PFS failure is a matter for the PFS to handle and is beyond the scope of the XDSM
25 standard. When the PFS fails, all non-persistent DMAPI resources are lost. The PFS is expected to clean up its own state when it is restarted. The DM application can then restart as well. Since sessions are not persistent, there is no need in this case for session recovery.

30 These two models do not describe all of the possible types of failure that may occur in a multi-node parallel

file system, as is used in preferred embodiments of the present invention. The multi-node system should also be capable of dealing with single-node failures, in which a file system instance or the PFS may fail on one or more
5 of the nodes, while continuing to work run on the others. In such a case, DMAPI 26 should also continue to operate and enable file access on the surviving nodes, while the PFS recovers on the failed nodes. A distributed DM application may likewise continue running on some nodes,
10 while other nodes (possibly including the session node for the DM application) have failed and are in the process of recovery.

Single-node failure may occur in the multi-node system either when a specific file system becomes
15 inaccessible on the node or when the entire PFS fails on the node. In the latter case, all file system instances that are managed by the PFS become inaccessible on that node. Handling of and recovery from a single node failure depend on whether the failed node is a source
20 node or a session node. When source node 42 fails (Fig. 2), events generated by that node become obsolete. If such events were already enqueued at session node 40, DM application 32 will continue to process the events. The processing may be unnecessary, since there is no longer
25 any file operation waiting for the response, but is harmless aside from the attendant loss in efficiency.

Session node failures are more difficult to handle. When a session node fails, all DMAPI resources, including all sessions, are lost on the failing node, although not
30 on other, surviving nodes. File operations on the surviving nodes may still be blocked, however, waiting for response from an event previously sent to a failed

session. It is therefore important to recover the session, possibly on another node, and to resume handling of pending events, so that file operations on the surviving nodes will be able to continue without failure.

5 Fig. 6 is a flow chart that schematically illustrates a method for dealing with session failure on node 40, in accordance with a preferred embodiment of the present invention. This type of failure scenario is not addressed by the XDSM standard. The present method is
10 needed to ensure that events generated at any of the nodes in cluster 20 can be handled in the proper manner, so that user threads that generated those events can accordingly be unblocked.

Session failure is detected at a session failure
15 step 110, preferably by a "heartbeat" or group service that checks connectivity in cluster 20, as is known in the art. Session manager (SM) 34 plays an important role during session recovery. When session node 40 fails, SM 34 is notified, at a notification step 112. The SM marks
20 the session as failed, but keeps all of the session details.

Recovery following the session failure is triggered by DM application 32, at a triggering step 114. There are two ways of triggering session recovery, depending on
25 whether the DM application itself has also failed, or only the PFS has failed:

- **Explicit recovery** - If the DM application failed at the same time as the session (due to node crash, for example), it must be restarted, possibly on
30 another node. The restarted DM application *explicitly* assumes the old session, using the DMAPI

function `dm_create_session()` and specifying the session ID. Assumption of the failed session triggers reconstruction of the session queue and the events on it on the new session node, as described below. The DM application can then continue handling events. If the DM application survives, it may notice that the session has failed, on account of the error codes that it receives in response to DMAPI function calls. In this case, the application may wait for the PFS to recover on the failed node, or it may alternatively move to another node and assume the failed session, as described above.

- **Implicit recovery** - Since the DM application executes as a separate process, independent of the PFS, it is possible that the PFS will recover before the DM application has even noticed the failure. In this case, session queue reconstruction is triggered *implicitly* when the DM application invokes any DMAPI function for the session at the session node. Explicit session assumption is unnecessary in this situation.

Calling `dm_create_session()` and supplying the session ID (explicit recovery) or calling any DMAPI function for the session (implicit recovery) causes DMAPI 26 on the new session node to contact SM 34, at a contact step 116. The SM records the new session node and session information string, if any, and changes the session state from failed to valid. It broadcasts the updated session details to all nodes, at a broadcast step 118.

While recovering the session, it is necessary to reconstruct the session queue at the new session node. Preferably, in order to reconstruct the session queue, session node 40 broadcasts a request to all of the nodes in cluster 20, at a request step 119. Upon receiving this request, the surviving nodes resubmit any pending synchronous events they may have to the new session node, at a resubmission step 120. This step causes the event tokens to be regenerated with the same IDs as they had before the failure. Certain events may not be recoverable by DMAPI 26 following a session failure. Asynchronous events are lost with no harm done. Events that originated from the failed session node, including user events, cannot be recovered by the resubmission mechanism described above.

Session failure results in the loss of resources associated with the events in the session queue, including DM access rights. These resources are not recovered simply by resubmitting the events. As a result, DMAPI functions may fail after session recovery due to invalid DM access rights. Furthermore, DMAPI 26 cannot determine after recovery which events were already being handled by the DM application prior to the failure, nor can it guarantee that none of the files in question were accessed or modified before the failure. All events resubmitted after a failure revert to the initial (non-outstanding) state. Similarly, when only a file system instance fails at the session node, all DM access rights for files in the file system are lost, although the events and tokens remain. After the file system instance is remounted, the DM application must reacquire the access rights. There is no guarantee that objects

have not been modified while the access rights were not held. Therefore, DM application 32 should be written so as to recover consistently from the loss of DM access rights, notwithstanding the associated loss of
5 information. For example, the DM application could be programmed to keep its own state of events in progress or to implement an appropriate consistency protocol, as is known in the art.

It is desirable to provide mechanisms that will
10 speed up recovery from a session failure and will prevent indefinite blocking of user applications. Preferably, if a failed session does not recover after a predetermined lapse of time, pending events are aborted at source node 42, and file operations associated with the events are
15 failed. User applications 30 can then retry the failed operations as appropriate.

It may also occur that SM 34 fails, in which case a new SM is appointed. The new SM must recover all of the information that was maintained by the previous SM. For
20 this purpose, the new SM preferably broadcasts a request to all nodes for session information. Each node responds by sending the SM a list of all the sessions existing on that node and the event dispositions for all the file systems that are mounted at the node. The new SM uses
25 this information to rebuild the collective session and disposition information.

The only information that may not be fully recovered in this manner is concerning sessions that existed on the failed SM node. As indicated above, each node preferably
30 keeps partial information on every session. Most of the missing information can therefore be retrieved locally at the new SM node, from the list of all sessions that was

maintained at that node. The only unrecoverable information regarding sessions that existed on the failed SM node is the session information string. As noted above, typically only a prefix of this string is maintained on nodes other than the session node itself.

The new SM reconstructs event dispositions based on information that it receives from the file system managers (FSMs) 36 in cluster 20. If a FSM has failed, the corresponding information is recovered from one of the other nodes that mounted the file system in question. If all of the nodes that mounted the file system have failed, however, the dispositions are lost, and new dispositions for the file system will have to be set when the file system is mounted on some node.

After reconstructing the dispositions, the new SM must make sure that the information it now has is consistent with all of the nodes. For this purpose, the SM sends its reconstructed dispositions to the FSMs, which in turn send it to all the nodes that have mounted corresponding file system instances. If some file system has no FSM (typically due to FSM failure), the dispositions are held at the SM and are sent to the new FSM when one is appointed.

Although preferred embodiments are described hereinabove with reference to a particular configuration of cluster 20 and parallel file system 28, it will be appreciated that the principles embodied in DMAPI 26 are similarly applicable in other parallel file system environments, as well. It will thus be understood that the preferred embodiments described above are cited by way of example, and that the present invention is not limited to what has been particularly shown and described

39878S5

hereinabove. Rather, the scope of the present invention includes both combinations and subcombinations of the various features described hereinabove, as well as variations and modifications thereof which would occur to
5 persons skilled in the art upon reading the foregoing description and which are not disclosed in the prior art.